Hierarchy in Fluid Construction Grammars

Joachim De Beule[1] and Luc Steels[1,2]

[1] Vrije Universiteit Brussel, Belgium,
[2] Sony Computer Science Laboratory, Paris, France
{joachim,steels}@arti.vub.ac.be,
WWW home page: http://arti.vub.ac.be/~{joachim,steels}/

Abstract. This paper reports further progress into a computational implementation of a new formalism for construction grammar, known as Fluid Construction Grammar (FCG). We focus in particular on how hierarchy can be implemented. The paper analyses the requirements for a proper treatment of hierarchy in emergent grammar and then proposes a particular solution based on a new operator, called the J-operator. The J-operator constructs a new unit as a side effect of the matching process.

1 Introduction

In the context of our research on the evolution and emergence of communication systems with human language like features[5], we have been researching a formalism for construction grammars, called Fluid Construction Grammar (FCG) [6]. Construction grammars (see [3], [2], [4] for introductions) have recently emerged from cognitive linguistics as the most successful approach so far for capturing the syntax-semantics mapping. In FCG (as in other construction grammars), a construction associates a semantic structure with a syntactic structure. The semantic structure contains various units (corresponding to lexical items or groupings of them) and semantic categorizations of these units. The syntactic structure contains also units (usually the same as the semantic structure) and syntactic categorizations.

FCG uses many techniques from formal/computational linguistics, such as feature structures for representing syntactic and semantic information and unification as the basic mechanism for the selection and activation of rules. But the formalism and its associated parsing and production algorithms have a number of unique properties: All rules are bi-directional so that the same rules can be used for both parsing and production, and they can be flexibly applied so that ungrammatical sentences or meanings that are only partly covered by the language inventory, can be handled without catastrophic performance degradation. We therefore prefer the term templates instead of rules. Templates have a score which reflects how 'grammatical' they are believed to be. The score is local to an agent and based solely on the agent's interaction with other agents. Our formalism is called *Fluid* Construction Grammar as it strives to capture the fluidity by which new grammatical constructions can enter or leave a language inventory.

The inventory of templates can be divided depending on what role they play in the overall language process. Morph(ological) templates and lex(ical)-stem templates license morphology and map lexical stems into partial meanings. Syn-cat and sem-cat templates identify syntactic or semantic features preparing or further instantiating grammatical constructions. Gram(matical) templates define grammatical constructions.

So far FCG has only dealt with single layered structures without hierarchy. It is obvious however that natural languages make heavy use of hierarchy: a sentence can contain a noun phrase, which itself can contain another noun phrase etc. This raises the difficult technical question how hierarchy can be integrated in FCG without loosing the advantages of the formalism, specifically the bi-directionality of the templates. This paper presents the solution to this problem that has proven effective in our computational experiments.

2 Hierarchy in Syntax

It is fairly easy to see what hierarchy means on the syntactic side and there is a long tradition of formalisms to handle it. To take an extremely simple example: the phrase "the big block" combines three lexical units to form a new whole. In terms of syntactic categories, we say that "the" is an article, "big" is an adjective, and "block" is a noun and that the combination is a noun phrase, which can function as a unit in a larger structure as in "the blue box next to the big block". Traditionally, this kind of phrase structure analysis is represented in graphs as in figure 1. Using the terminology of (fluid) construction grammar,

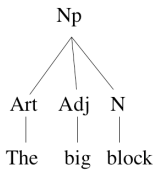

Fig. 1. Hierarchical syntactic structure for a simple noun phrase

we would say that "the big block" is a noun phrase construction of the type Art-Adj-Noun. It consists of a noun-phrase (NP) unit which has three subunits, one for the Article, one for the Adjective and one for the Noun. "The blue box next to the big block" is another noun phrase construction of type NP-prep-NP. So constructions come in different types and subtypes. The units in a construction participate in particular grammatical relations (such as head, complement, determiner, etc.) but the relational view will not be developed here.

Various syntactic constraints are associated with the realization of a construction which constrain its applicability. For example, in the case of the Art-

Adj-Noun construction, there is a particular word order imposed, so that the Article comes before the Adjective and the Adjective before the Noun. Agreement in number between the article and the noun is also required. In French, there would also be agreement between the adjective and the noun, and there would be different Art-Adj-Noun constructions with different word order patterns depending on the type of adjective and/or its semantic role ("une fille charmante" vs. "un bon copain").

When a parent-unit is constructed that has various subunits, there are usually properties of the subunits (most often the head) that are carried over to the parent-unit. For example, if a definite article occurs in an Art-Adj-Noun construction, then the noun phrase as a whole is also a definite noun phrase. Or if the head noun is singular than the noun phrase as a whole is singular as well. It follows that the grammar must be able to express (1) what kind of units can be combined into a larger unit, and (2) what the properties are of this larger unit.

The first step toward hierarchical constructions consists in determining what the syntactic structure looks like before and after the application of the construction. Right before application of the construction in parsing, the syntactic structure is as shown in figure 2. It corresponds to the left part of figure 6. Application of the Art-Adj-Noun construction to this syntactic structure should

```
((top
  (syn-subunits (determiner modifier head))
  (form ((precedes determiner modifier)
         (precedes modifier head))))
 (determiner
   (syn-cat ((lex-cat article) (number singular)))
   (form ((stem determiner "the"))))
 (modifier
   (syn-cat ((lex-cat adjective)))
   (form ((stem modifier "big"))))
 (head
   (syn-cat ((lex-cat noun) (number singular)))
   (form ((stem head "block")))))
```

Fig. 2. Syntactic structure for the phrase "The big block" as it is represented in FCG before the application of an NP construction of the type Art-Adj-Noun. The structure contains four units, one of them (the top unit) has the other three as subunits. The lexicon contributes the different units and their stems. The precedes-constraints are directly detectable from the utterance. This structure should trigger the NP construction.

result in the structure shown in figure 3. It corresponds to the right part of figure 6. As can be seen a new NP-unit is inserted between the top-unit and the other units. The NP-unit has the determiner, modifier and head-unit as subunits

```
((top
    (syn-subunits (np-unit)))
 (np-unit
    (syn-subunits (determiner modifier head))
    (form ((precedes determiner modifier)
           (precedes modifier head)))
    (syn-cat ((lex-cat NP) (number singular))))
 (determiner
    (syn-cat ((lex-cat article) (number singular)))
    (form ((stem determiner "the"))))
 (modifier
    (syn-cat ((lex-cat adjective)))
    (form ((stem modifier "big"))))
 (head
    (syn-cat ((lex-cat noun) (number singular)))
    (form ((stem head "block")))))
```

Fig. 3. Syntactic structure for the phrase "The big block" as it is represented in FCG after the application of an NP construction of the type Art-Adj-Noun. A new NP-unit is inserted between the top-unit and the other units, which now covers the article, adjective and noun units.

and also contains the precedence constraints involving these units. The lexical category of the new unit is NP and it inherits the number of the head-unit. Let us now investigate what the constructions look like that license this kind of transformation.

In FCG a construction is defined as having a semantic pole (left) and a syntactic pole (right). The application of a template consists of a matching phase followed by a merging phase. In parsing, the syntactic pole of a construction template must match with the syntactic structure after which the semantic pole may be merged with the semantic structure to get the new semantic structure. In production, the semantic pole must match with the semantic structure and the syntactic pole is then merged with the syntactic structure. Merging is not always possible. This section considers first the issue of syntactic parsing.

Our key idea to handle hierarchy is to construct a new unit as a side effect of the matching and merging processes. Specifically, we can assume that, for the example under investigation, the syntactic pole should at least contain the units shown in figure 4. This matches with the syntactic structure given in 2 and can therefore be used in parsing to test whether a noun-phrase occurs. But it does not yet create the noun-phrase unit.

This is achieved by the J-operator.[3] Units marked with the J-operator are ignored in matching. When matching is successful, the new unit is introduced and bound to the first argument of the J-operator. The second argument should

[3] J denotes the first letter of Joachim De Beule who established the basis for this operator.

```
((?top
    (syn-subunits (== ?determiner ?modifier ?head))
    (form (== (precedes ?determiner ?modifier)
              (precedes ?modifier ?head))))
 (?determiner
    (syn-cat (== (lex-cat article) (number singular))))
 (?modifier
    (syn-cat (== (lex-cat adjective))))
 (?head
    (syn-cat (== (lex-cat noun) (number singular)))))
```

Fig. 4. The part of the NP-construction that is needed to match the structure in figure 2. Symbols starting with a question-mark are variables that get bound during a successful match or merge. The ==, or *includes symbol*, specifies that the following list should at least contain the specified elements but possibly more.

already have been bound by the matching process to the parent unit from which the new unit should depend. The third argument specifies the set of units that will be pulled into the newly created unit. The new unit can contain additional slot specifications, specified in the normal way, and all variable bindings resulting from the match are still valid. An example is shown in figure 5.

```
((?top
    (syn-subunits (== ?determiner ?modifier ?head))
    (form (== (precedes ?determiner ?modifier)
              (precedes ?modifier ?head))))
 (?determiner
    (syn-cat (==1 (lex-cat article) (number ?number))))
 (?modifier
    (syn-cat (==1 (lex-cat adjective))))
 (?head
    (syn-cat (==1 (lex-cat noun) (number ?number))))
 ((J ?new-unit ?top (?determiner ?modifier ?head))
    (syn-cat (np (number ?number)))))
```

Fig. 5. The pole of figure 4 extended with a J-operator unit. The function of the ==1 or *includes-uniquely* symbol will be explained shortly, in matching it roughly behaves like the == symbol.

Besides creating the new unit and adding its features, the overall structure should change also. Specifically, the new-unit is declared a subunit of its second argument (i.e. ?top in figure 5) and all feature values specified in this parent unit are moved to the new unit (i.e. the `syn-subunits` and `form` slots in figure 5.) This way the precedence relations in the form-slot of the original parent or any other categories that transcend single units (like intonation) can automatically

be moved to the new unit as well. Thus the example syntactic structure of figure 2 for "the big block" is indeed transformed into the target structure of figure 3 by applying the pole of figure 5. The operation is illustrated graphically in figure 6. Note that now the np-unit is itself a unit ready to be combined with others if necessary.

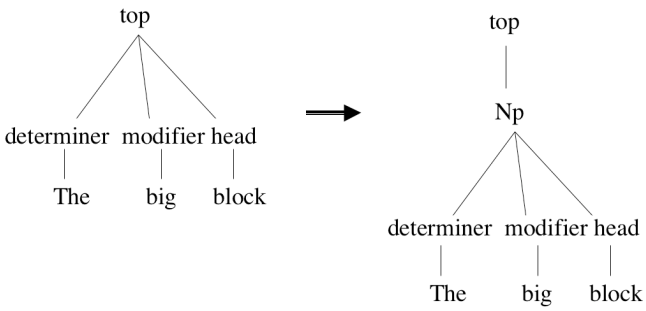

Fig. 6. Restructuring performed by the J-operator

Formally, the J-operator is a tertiary operator and is written as (variable names are arbitrary):

```
(J ?new-unit ?parent (?child-1 ... ?child-n))
```

The second argument is called the parent argument of the J-operator and the third the children-argument. Any unit-name in the left or right pole of a template may be governed by a J-operator. The parent and children-arguments should refer to other units in the same pole.

Matching a pattern containing a J-operator against a target structure is equivalent to matching the pattern *without* the unit marked by the J-operator.

Assume a set of bindings for the parent and children in the J-operator, then merging a pattern containing a J-operator with a target is defined as follows:

1. Unless a binding is already present, the set of bindings is extended with a binding of the first argument `?new-unit` of the J-operator to a new constant unit-name `N`.
2. A new pattern is created by removing the unit marked by the J-operator from the original pattern and adding a unit with name `N`. This new unit has as slots the union of the slots of the unit marked by the J-operator and of the unit in the original pattern as specified by the parent argument. This pattern is now merged with the target structure.
3. The feature values of the parent specified in the J-pattern are removed from the corresponding unit in the target structure and the new unit is made a subunit of this unit.
4. Finally, all remaining references to units specified by the children are replaced by references to the new unit.

This is illustrated for the target structure in figure 2 and the pattern of figure 5. The pattern figure 5 matches with figure 2 because the J-unit is ignored. The resulting bindings are:

```
[?top/top, ?determiner/determiner, ?modifier/modifier,
 ?head/head, ?number/singular]                              (B1)
```

The new pattern is given by

```
((?top
    (syn-subunits (== ?determiner ?modifier ?head))
    (form (== (precedes ?determiner ?modifier)
              (precedes ?modifier ?head))))
 (?determiner
    (syn-cat (==1 (lex-cat article) (number ?number))))
 (?modifier
    (syn-cat (==1 (lex-cat adjective))))
 (?head
    (syn-cat (==1 (lex-cat noun) (number ?number))))
 (np-unit
    (syn-cat (np (number ?number)))
    (syn-subunits (== ?determiner ?modifier ?head))
    (form (== (precedes ?determiner ?modifier)
              (precedes ?modifier ?head)))))
```

Before merging, these bindings are extended with a binding `[?new-unit/np-unit]`. Merging this pattern with figure 2 given the bindings (B1) results in

```
((top
    (syn-subunits (determiner modifier head))
    (form ((precedes determiner modifier)
           (precedes modifier head)))
 (np-unit
    (syn-subunits (determiner modifier head))
    (form ((precedes determiner modifier)
           (precedes modifier head)))
    (syn-cat ((lex-cat NP) (number singular)))))
 (determiner
    (syn-cat ((lex-cat article) (number singular)))
    (form ((stem determiner "the"))))
 (modifier
    (syn-cat ((lex-cat adjective)))
    (form ((stem modifier "big"))))
 (head
    (syn-cat ((lex-cat noun) (number singular)))
    (form ((stem head "block")))))
```

The last two steps remove the syn-subunits and form slots from the top unit and result in the structure of figure 3.

Note that the application of a template now actually consists of three phases: first the matching of a pole against the target structure, next the merging of the *same* structure with the altered pole as specified above and finally the normal merging phase between the template's other pole and the corresponding target structure (although here too J-operators might be involved).

In syntactic production the syntactic structure after lexicon-lookup and before application of the Art-Adj-Noun construction is as shown in figure 7. As will

```
((top
    (syn-subunits (determiner modifier head))
 (determiner
    (form ((stem determiner "the"))))
 (modifier
    (form ((stem modifier "big"))))
 (head
    (form ((stem head "block"))))))
```

Fig. 7. Syntactic structure while producing "the big block", before the application of an NP-construction.

be discussed shortly, in production the left (semantic) pole of the construction is matched with the semantic structure and its right (syntactic) pole (shown in figure 5) has to be *merged* with the above syntactic structure. Here again the J-operator is used to create a new unit and link it from the top, yielding again the structure shown in figure 3.

It is important to realise that merging may fail (and hence the application of the construction may fail) if there is an incompatibility between the slot-specification in the template's pattern and the target structure with which the pattern is merged. Indeed this is the function of the *includes-uniquely* symbol `==1`. It signals that there can only be one clause with the predicates that follow. Whereas if the slot-specification starts with a normal includes symbol `==`, it means that the clause(s) can simply be added. The different forms of a slot-specification and their behavior in matching and merging are summarised in the table below:

Notation	*Read as*	*Matching and merging behavior*
$(a_1...a_n)$	equals	target must contain *exactly* the given elements
$(== a_1..a_n)$	includes	target must contain *at least* the given elements
$(== 1\ a_1...a_n)$	includes uniquely	target must contain *at least* the given elements and may not contain duplicate predicates

For example, the syn-cat slot of the determiner unit in the NP-construction's syntactic pole (see figure 5) is specified as `(==1 (lex-cat article) (number ?number))`. This can be merged only if the lex-cat of the target unit is none other than article and if the `?number` variable is either not yet bound (in which case it becomes

bound to the one contained in the target unit) or else only if it is bound to the number value in the target unit.

Note also that no additional mechanisms are needed to implement agreement or the transfer from daughter to parent. Indeed, in figure 5 `?number` is the same for the determiner and modifier units. If these units would have different number values in the target structure then the merge is blocked. This implements a test on agreement in number. And because the new-unit has its number category specified by the same variable `?number`, it will inherit the number of its constituents as specified.

provided, in which Art-Adj-Noun specifies that

3 Hierarchy in Semantics

The mechanisms proposed so far implement the basic mechanism of syntactically grouping units together in more encompassing units. Any grammar formalism supports this kind of grouping. However constructions have both a syntactic and a semantic pole, and so now the question is how hierarchy is to be handled semantically in tight interaction with syntax.

A construction such as "the big block" not only simply groups together the meanings associated with its component units but also adds additional meaning, including what should be done with the meanings of the components to obtain the meaning of the utterance. FCG uses a Montague-style semantics (see [1] for an introduction), which means that individual words like "ball" or "big" introduce predicates and that constructions introduce second order semantic operators that say how these predicates are to be used (see [7] for a sketchy description of the system IRL generates this kind of operators and how it is used).

For example, we can assume a semantic operator `find-referent-1`, which is a particular type of a find-referent operator that uses a quantifier, like `[the]`, a property, like `[big]`, and a prototype of an object, like `[ball]`, to find an object in the current context. It would do this by filtering all objects that match the prototype, then filtering them further by taking the one who is biggest, and then taking out the unique element of the remaining singleton.

The initial semantic structure containing the meaning of the phrase "the big block" now looks as in figure 8. Lexicon-lookup introduces units for each of

```
((top
   (meaning ((find-referent-1 obj det1 prop1 prototype1)
             (quantifier det1 [the])
             (property prop1 [big])
             (prototype prototype1 [ball])))))
```

Fig. 8. The initial semantic structure for producing the phrase "the big block"

the parts of the meaning that are covered by lexical entries and transforms this structure into the one in figure 9: The challenge now is to apply a construction

```
((top
    (sem-subunits (determiner modifier head))
    (meaning ((find-referent-1 obj det1 prop1 prototype1))))
 (determiner
    (referent det1)
    (meaning ((quantifier det1 [the]))))
 (modifier
    (referent prop1)
    (meaning ((property prop1 [big]))))
 (head
    (referent prototype1)
    (meaning ((prototype prototype1 [ball])))))
```

Fig. 9. The semantic structure for producing "the big block" after lexicon lookup. Three lexical entries were used, respectively covering the parts of meaning `(quantifier det1 [the])`, `(property prop1 [big])` and `(prototype prototype1 [ball])`.

which pulls out the additional part of the meaning that is not yet covered and constructs the appropriate new unit. Again the J-operator can be used with exactly the same behavior as seen earlier. The J-unit is ignored during matching but introduces a new unit during merging as explained before: the new unit is linked to the parent and the parent slot specifications contained in the template are moved to the new unit.

Hence, the semantic pole of the Art-Adj-Noun construction looks as follows:

```
((?top
    (sem-subunits (== ?determiner ?modifier ?head))
    (meaning
      (== (find-referent-1 ?obj ?det1 ?prop1 ?prototype1))))
 (?determiner (referent ?det1))
 (?modifier (referent ?prop1))
 (?head (referent ?prototype1))
 ((J ?new-unit ?top) (referent ?obj)
  (?determiner ?modifier ?head)))
```

Application of this semantic pole to the semantic structure of figure 9 yields:

```
((top
   (sem-subunits (np-unit)))
 (np-unit
   (referent obj)
   (sem-subunits (determiner modifier head))
   (meaning ((find-referent-1 obj det1 prop1 prototype1))))
```

```
(determiner
   (referent det1)
   (meaning ((quantifier det1 [the]))))
(modifier
   (referent prop1)
   (meaning ((property prop1 [big]))))
(head
   (referent prototype1)
   (meaning ((prototype prototype1 [ball]))))
```

As on the syntactic side, various kinds of selection restrictions could be added. On the semantic side they take the the form of semantic categories that may have been inferred during re-conceptualization (i.e. using sem-templates.) These selection restrictions would block the application of the construction while matching in production and while merging in parsing. For example, we could specialise the construction's semantic pole to be specific to the domain of physical objects (and hence physical properties and prototypes of physical objects) as follows:

```
((?top
    (sem-subunits (== ?determiner ?modifier ?head))
    (meaning
      (==
       (find-referent-1 ?obj ?quantifier ?property ?prototype))))
 (?determiner (referent ?quantifier))
 (?modifier
    (referent ?property)
    (sem-cat (==1 (property-domain ?property physical))))
 (?head
    (referent ?prototype)
    (sem-cat (==1 (prototype-domain ?prototype physical))))
 ((J ?new-unit ?top)
    (referent ?obj)
    (sem-cat (==1 (object-domain ?obj physical)))))
```

The complete Art-Adj-Noun construction as a whole is shown in figure 10.

syntactic parsing. syntactic pole. This unit is top unit are

As an example of semantic parsing, suppose the semantic structure after lexicon-lookup and semantic categorisation is as in figure 11.

Merging now takes place and succeeds because all semantic categories are compatible. The final result is:

```
((top
    (sem-subunits (np-unit)))
 (np-unit
    (referent ?obj)
    (sem-subunits (determiner modifier head))
    (meaning ((find-referent-1 ?obj ?quant1 ?prop1 ?prototype1)))
```

```
Art-Adj-Noun Construction:
     ((?top
         (sem-subunits (== ?determiner ?modifier ?head))
         (meaning
           (==
            (find-referent-1 ?obj ?quantifier ?property ?prototype))))
      (?determiner (referent ?quantifier))
      (?modifier
         (referent ?property)
         (sem-cat (==1 (property-domain ?property physical))))
      (?head
         (referent ?prototype)
         (sem-cat (==1 (prototype-domain ?prototype physical))))
      ((J ?new-unit ?top)
         (referent ?obj)
         (sem-cat (==1 (object-domain ?obj physical)))))
     <-->
     ((?top
         (syn-subunits (== ?determiner ?modifier ?head))
         (form (== (precedes ?determiner ?modifier)
                   (precedes ?modifier ?head))))
      (?determiner
         (syn-cat (==1 (lex-cat article) (number ?number))))
      (?modifier
         (syn-cat (==1 (lex-cat adjective))))
      (?head
         (syn-cat (==1 (lex-cat noun) (number ?number))))
      ((J ?new-unit ?top)
         (syn-cat (np (number ?number)))))
```

Fig. 10. The complete Art-Adj-Noun construction. The double arrow $< -- >$ seperates the semantic pole of the construction (above the arrow) form the syntactic pole (below the arrow) and suggests the bi-directional application of the construction during parsing and producing.

```
((top
   (sem-subunits (determiner modifier head)))
 (determiner (referent ?quant1)
   (meaning ((quantifier ?quant1 [the]))))
 (modifier
   (referent ?prop1)
   (meaning ((property ?prop1 [big])))
   (sem-cat ((property-domain ?prop1 physical-object))))
 (head (referent ?prototype1)
       (meaning ((prototype ?prototype1 [ball])))
       (sem-cat ((prototype-domain ?prototype1 physical-object)))))
```

Fig. 11. Semantic structure after lexicon lookup and semantic categorisation while parsing the phrase "the big block".

```
      (sem-cat ((object-domain ?obj physical))))
   (determiner
      (referent ?det1)
      (meaning ((quantifier ?quant1 [the]))))
   (modifier
      (referent ?prop1)
      (meaning ((property ?prop1 [big])))
      (sem-cat ((property-domain ?prop1 physical-object))))
   (head
      (referent ?prototype1)
      (meaning ((prototype ?prototype1 [ball])))
      (sem-cat ((prototype-domain ?prototype1 physical-object)))))
```

4 The J-operator in lexicon look-up

In the previous sections it was assumed that the application of lex-stem templates resulted in a structure containing separate units for every lexical entry. For example, it was assumed that the lex-stem templates transform the flat structure of figure 8 into the one in figure 9. However an additional transformation is needed for this, and in this section we show how the the J-operator can be used to accomplish this.

In production the initial semantic structure is as in figure 8. Now consider the following lexical entry:

```
(def-lex-template [the]
    ((?top (meaning (== (quantifier ?x [the]))))
     ((J ?new-unit ?top)))
  <-->
  ((?top (syn-subunits (== ?new-unit))
         (form (== (string ?new-unit "the"))))
   ((J ?new-unit ?top)
         (form (== (stem ?new-unit "the")))
         (syn-cat (== (lex-cat article))))))),
```

and similar entries for `[big]` and `[block]`.

Matching the left pole of the `[the]`-template with the structure in figure 8 yields the bindings:

```
[?top/top, ?x/det1].
```

As explained, merging this pole with figure 8 extends these bindings with the binding `[?new-unit/[the]-unit]` and yields:

```
((top
   (sem-subunits ([the]-unit))
   (meaning ((find-referent-1 obj det1 prop1 prototype1)
             (property prop1 [big])
```

```
                (prototype prototype1 [ball])))
  ([the]-unit
    (referent det1)
    (meaning ((quantifier det1 [the])))))
```

Note that, because of the working of the J-operator, a new unit is created which has absorbed the `(quantifier det1 [the])` part of the meaning from the top unit. Applying also the templates for `[big]` and `[block]` finally results in the desired structure shown in figure 9.

In production, the initial syntactic structure is empty and simply equal to `((top))`. But merging the right pole of the [the]-template given the extended binding yields:

```
((top
   (syn-subunits ([the]-unit)))
 ([the]-unit
   (form ((string [the]-unit "the") (stem [the]-unit "the")))
   (syn-cat ((lex-cat article)))))
```

Similarly applying the templates for `[big]` and `[block]` results in figure 7.

As a final example, consider that the initial semantic structure in parsing is also empty and equal to `((top))`. Merging of the left poles of the lexical entries results in

```
((top
   (sem-subunits ([the]-unit [big]-unit [block]-unit)))
 ([the]-unit (referent ?x-1)
   (meaning ((quantifier ?x-1 [the]))))
 ([big]-unit
   (referent ?x-2)
   (meaning ((property ?x-2 [big]))))
 ([block]-unit
   (referent ?x-3)
   (meaning ((prototype ?x-3 [ball])))))
```

which is, apart from the sem-cat features which have to be added by sem-templates (re-conceptualization), indeed equivalent with figure 11.

5 Conclusions

This paper introduced the J-operator and showed that it is a very elegant solution to handle hierarchical structure, both in the syntactic and semantic domain. All the properties of FCG, and particularly the bi-directional applicability of templates, could be preserved. There are of course many related topics and remaining issues that could not be discussed in the present paper. One issue is the percolation of properties from the 'head' node of a construction to the governing node. Although this can be regulated explicitly (as shown in the Art-Adj-Noun

construction where the number of the noun percolates to the number of the governing NP), it has been argued (in HPSG related formalisms) that there are default percolations. This can easily be implemented in FCG as will be shown in a forthcoming paper. Another issue is the handling of optional components, i.e. elements of a construction which do not obligatory appear. This can be handled by another kind of operator that regulates which partial matches are licenced as will also be shown in another paper. Finally, we have already implemented learning operators that invent or acquire the kind of constructions discussed here, but their presentation goes beyond the scope of the present paper.

6 Acknowledgement

This research has been conducted at the Sony Computer Science Laboratory in Paris and the University of Brussels VUB Artificial Intelligence Laboratory. It is partially sponsored by the ECAgents project (FET IST-1940). We are indebted to stimulating discussions about FCG with Benjamin Bergen, Joris Bleys, and Martin Loetzsch.

References

1. Dowty, D., R. Wall, and S. Peters (1981) Introduction to Montague Semantics. D. Reidel Pub. Cy., Dordrecht.
2. Goldberg, A.E. 1995. *Constructions.: A Construction Grammar Approach to Argument Structure.* University of Chicago Press, Chicago
3. Kay, P. and C.J. Fillmore (1999) Grammatical Constructions and Linguistic Generalizations: the What's X Doing Y? Construction. Language, may 1999.
4. Michaelis, L. 2004. *Entity and Event Coercion in a Symbolic Theory of Syntax* In J.-O. Oestman and M. Fried, (eds.),Construction Grammar(s): Cognitive Grounding and Theoretical Extensions. Constructional Approaches to Language series, volume 2. Amsterdam: Benjamins.
5. Steels, L. (2003) Evolving grounded communication for robots. Trends in Cognitive Science. Volume 7, Issue 7, July 2003 , pp. 308-312.
6. Steels, L. (2004) Constructivist Development of Grounded Construction Grammars Scott, D., Daelemans, W. and Walker M. (eds) (2004) Proceedings Annual Meeting Association for Computational Linguistic Conference. Barcelona. p. 9-1
7. Steels, L. (2000) The Emergence of Grammar in Communicating Autonomous Robotic Agents. In: Horn, W., editor, Proceedings of European Conference on Artificial Intelligence 2000. Amsterdam, IOS Publishing. pp. 764-769.